



# Programming with Android: Geo-localization and Google Maps Services

**Federico Montori**

**Dipartimento di Scienze dell'Informazione**

**Università di Bologna**



# Outline

## Geo-localization techniques

Location **L**istener and Location **M**anager

**Google Maps Library:** *Getting an activation key*

**Google Maps Library:** *Display a Map*

**Google Maps Library:** *Adjust the Camera*

**Google Maps Library:** *Manage events*

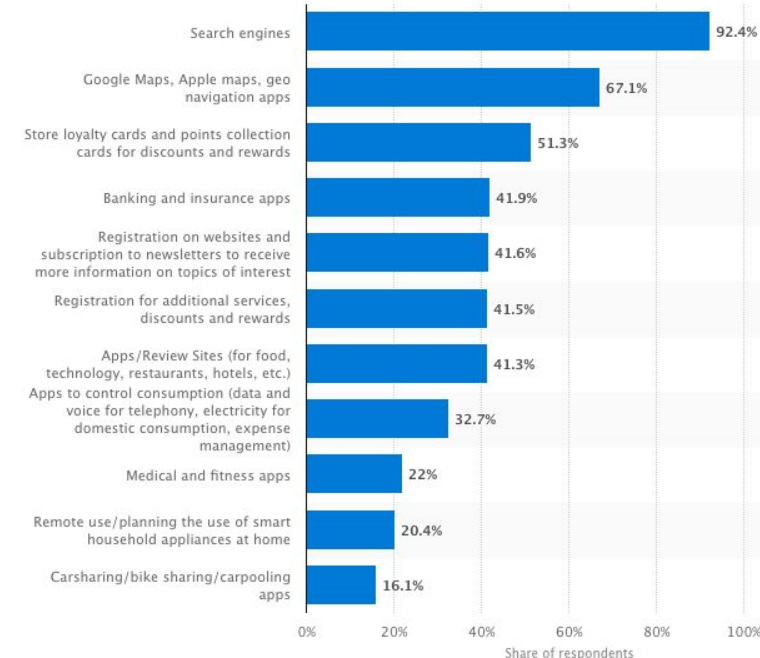
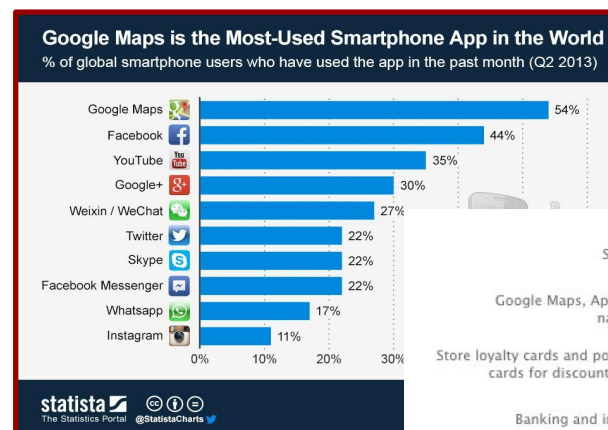
**Google Maps Library:** *Manage overlays*



# Android: Localization basics ...

□ **Geolocalization** □ Identification of the real-world geographic location of an the end-user.

- ✧ Feature supported by several Android applications.
- ✧ One of the reason of the popularity of today's smartphone devices.
- ✧ Made possible by the combination of **hardware radio transceivers** and **software localization algorithms**.

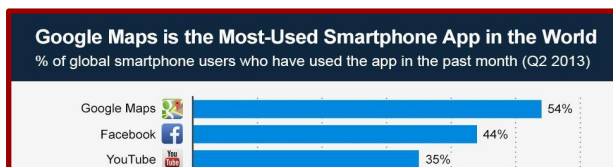




# Android: Localization basics ...

□ **Geolocation** □ Identification of the real-world geographic location of an the end-user.

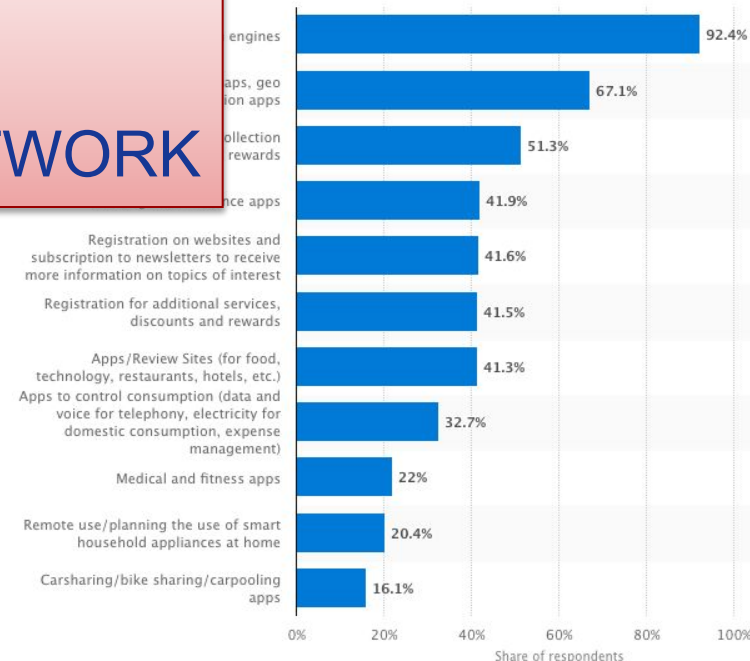
✦ Feature supported by several Android applications



✦ LOCALIZATION THROUGH GPS  
✦ LOCALIZATION THROUGH WI-FI  
✦ LOCALIZATION THROUGH CELLULAR NETWORK

devices.

✦ Made possible by the combination of **hardware radio transceivers** and **software localization algorithms**.





# Context Awareness



Where we are

What we are doing



Context Aware Computing is the possibility for a system to make its computation dependent on the context.



With whom we are



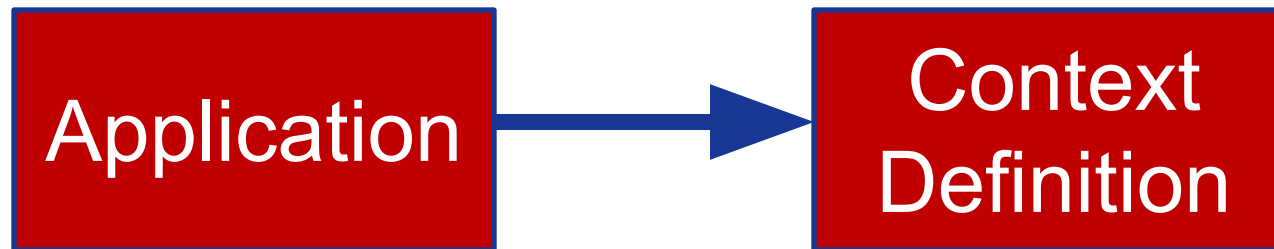
Our activity

What time is it



# Context Awareness

- ❖ Context may not be unique
  - For some applications it may be “Alice is running in the park alone”
  - Other may focus on different aspects “Alice has her phone running out of battery and is 5km away from her car”

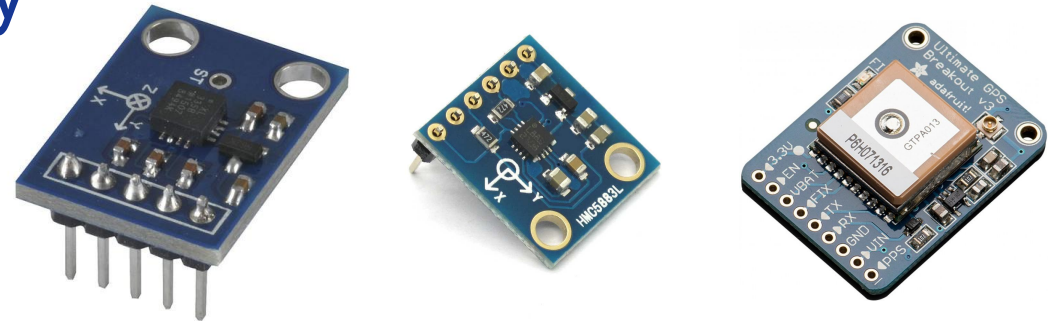


*“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”  
(Dey, Abowd 1999)*

# Context types

## ❖ Context can be either Primary

- If it is defined as raw data
  - Sensors, GPS, time



## ❖ Or it can be secondary

- If some form of data fusion has been performed
  - Calculate the season
  - Identify a face
  - ...





# Context **LifeCycle**

## What to do with context

Inferred context may be useful for **other service**

**Learning and deriving** context from extracted data.

**Machine learning** and rule based systems.

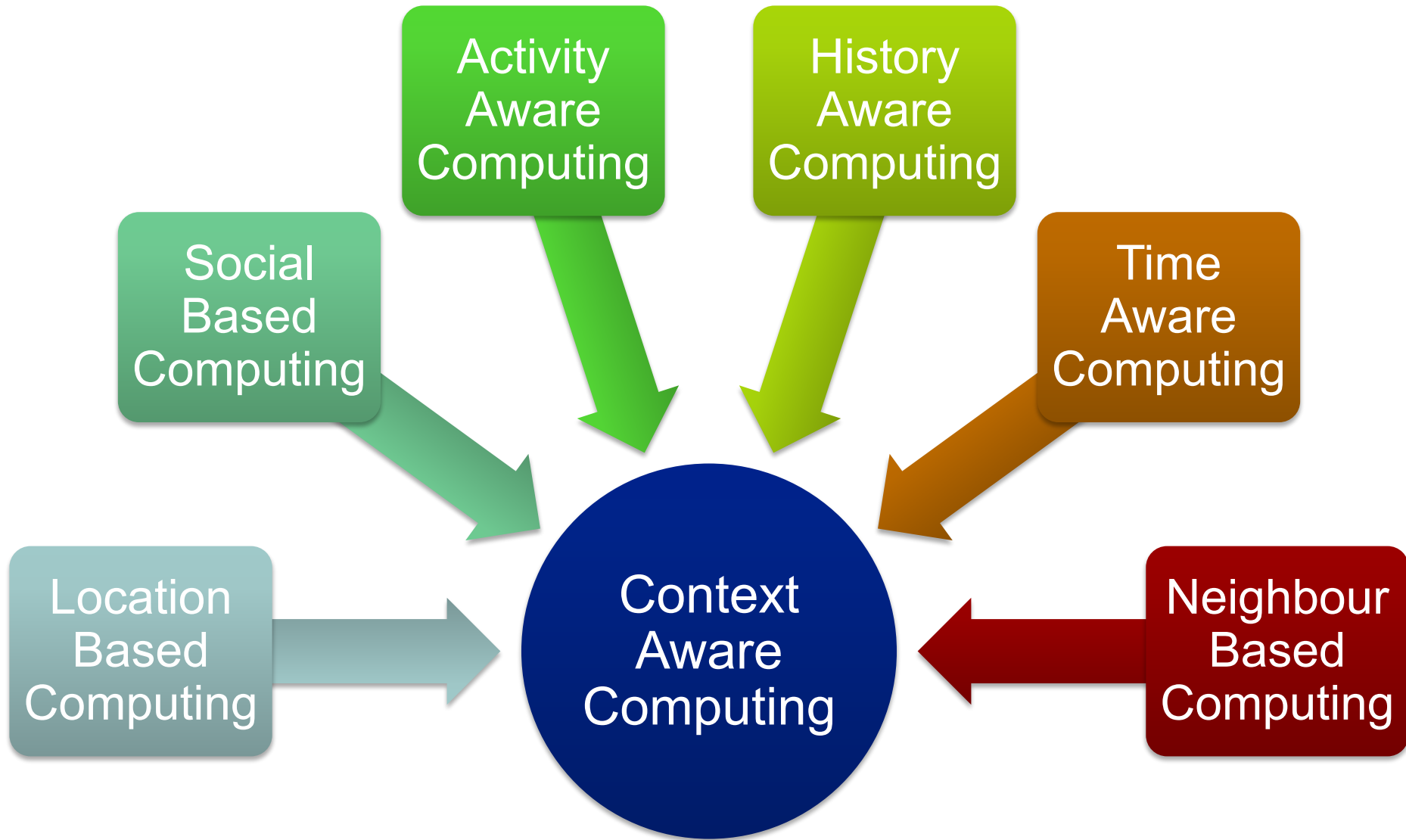
Gathering can be performed by **reading sensor values**, or getting information from social networks

How to extract context from raw data. Several possibilities (**graph based, ontologies, manual, ...**)





# Context Aware **Systems**





# A popular example: IFTTT

## ❖ If This Then That (IFTTT)

- Define rules (recipes) to perform action depending on previous states (context)
- ❖ Community based
- ❖ Integrated with external devices (location, buttons, ...)





# Other examples

## ❖ I search “Milano”

- If I am close to Milano, I may be looking for the city
- If I am close to Via Milano in Rome, I may be looking for it

## ❖ So Context Awareness based on what?

- Sky's the limit:
  - Geolocation data, Calendar Events, Neighbors, Activity recognition, Previous Events, External Events, Running pace, ...



# Some use cases

- ❖ E-health
  - Monitoring of patients
- ❖ Proximity marketing
  - Discounts on watched products
- ❖ Networking
  - Dynamic Adaptive Video Streaming
- ❖ IoT
  - Perform actions on closeness to other devices



# And some **challenges**

## ❖ Battery

- No one would use a system if it depletes the battery
- Keep in mind that most of the computation is performed on mobile devices

## ❖ Context Definition

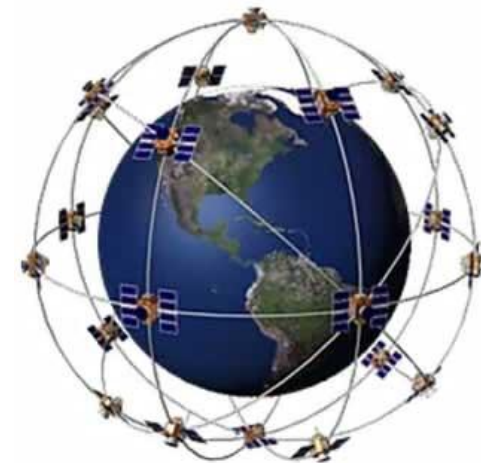
- Has to be defined per-scenario
- How to generalize?
- Liability?



# Android: **Localization basics ...**

GPS stands for **Global Positioning System**

- Fleet of satellites orbiting at a height of 20000km.
- Fleet composed of 24/32 operative satellites.
- Orbit period of 12 hours, speed of 3.9 Km/s.



Navigation systems available:

❖ **Navstar** → operated by the US Department of Defence (DoD)  
for civil/military applications

❖ **Glonass** → operated by the Russian Defence Forces.

❖ **Galileo** → operated by the EU

❖ **Beidou** → operated by China

❖ **NavIC** → operated by India

❖ **QZSS** → operated by Japan



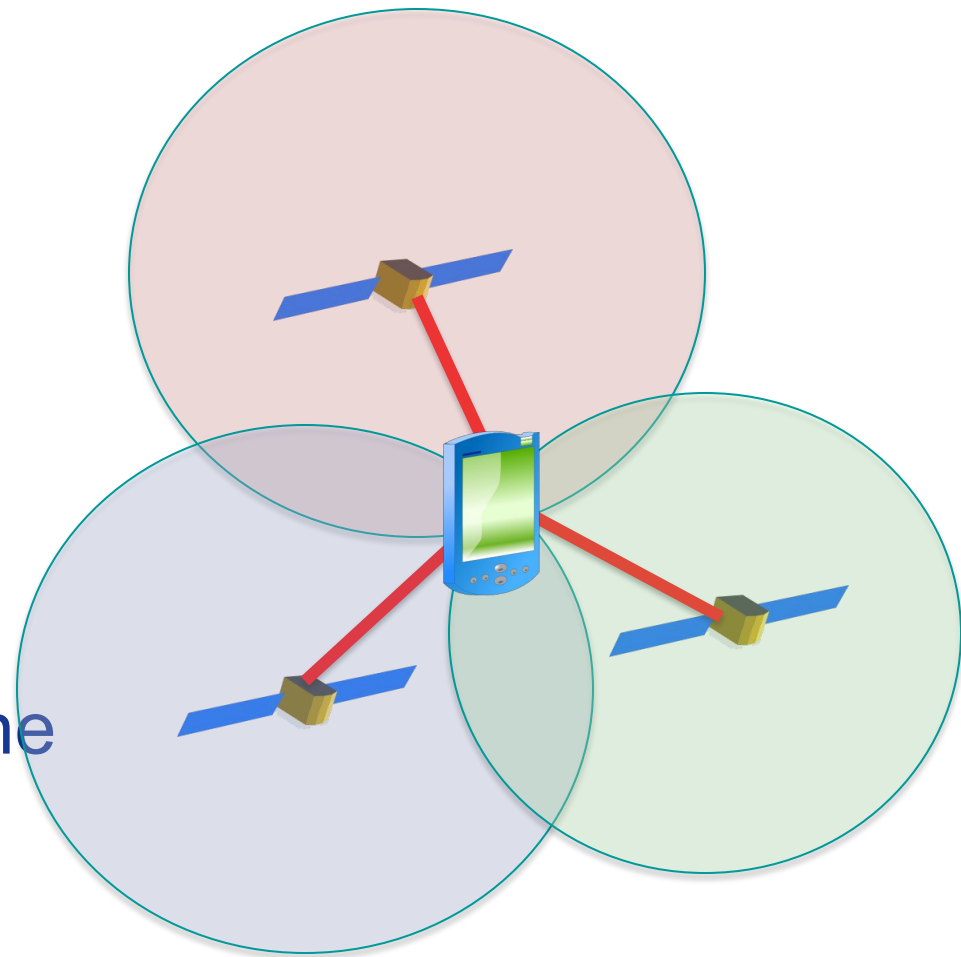
# Android: Localization basics ...

Each satellite sends periodically:

- Its current **location**
- Current **time** of the day (atomic clock)

**GPS receiver operations:**

1. Passively receive data (no transmit)
2. Compute delay of received signal
3. From delay compute the distance to the satellite (distance = delay \* c)
4. From multiple distance (at least 3), determine current locations.





# Android: Localization basics ...

**PROBLEM:** In order to calculate delay of received signal, the end-user clock must be synchronized with the satellite clock...

## SOLUTION

- ✧ Utilize four satellite instead of three (minimum)
- ✧ GPS receiver solves a system with four unknown variables

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = ([\tilde{t}_r + b - t_i]c)^2, \quad i = 1, 2, \dots, n$$

$x_i, y_i, z_i \rightarrow$  user's location

$b \rightarrow$  user clock skew





# Android: **Localization basics ...**

Each satellite transmits on two **frequencies** in the UHF band:

- ✦ L1 channel □ **civilian** data
- ✦ Signals encoded using code division multiple access (**CDMA**)
- ✦ Together with data/location, each satellite transmits the **almanac** data, i.e. orbital courses of the satellites.
- ✦ Through the almanac, GPS receiver knows about satellites visible at its location.



# Android: **Localization basics ...**

Wi-Fi Localization is performed through triangulation or through **radio fingerprinting** approach (the latter used by Android):

1. Smartphone turns on the WiFi interface, and detects MAC and SSID of WiFi routers in its range.
2. Smartphone makes a query to the Google location service.
3. Based on stored information about known WiFi networks, Google provides hints about current location.

**Q. HOW is the Google database populated?**

**A. By users, enabling the Google's location service.**

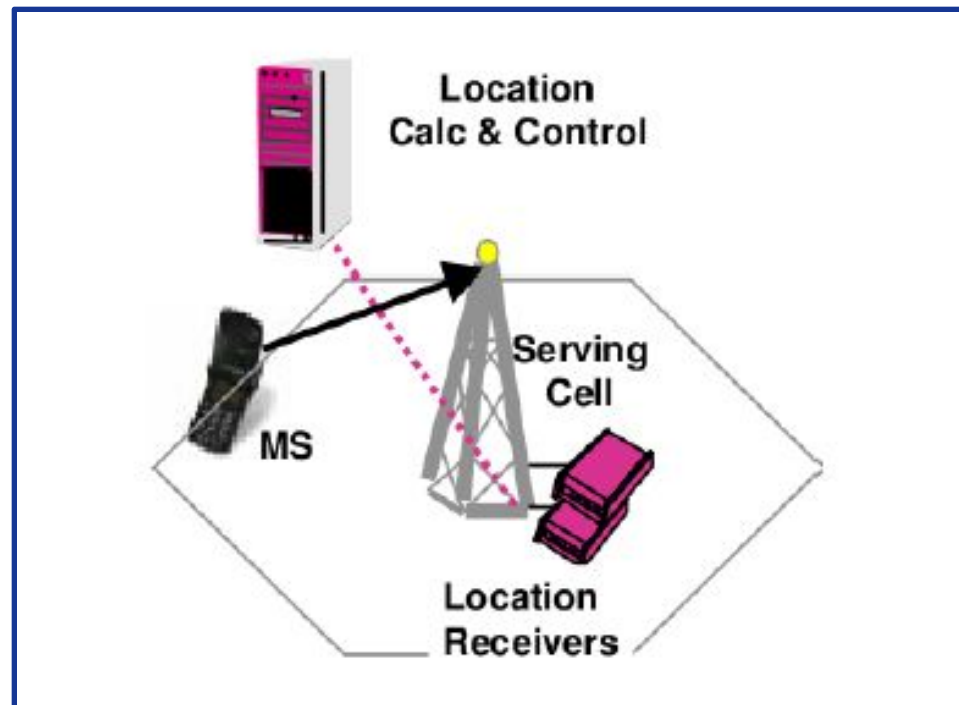




# Android: Localization basics ...

✦ Cellular Localization is performed by recognizing the **mobile cell tower** which the smartphone is attached to. HOW?

✦ Similar to previous case, current location is determined on the basis of the ID of the cellular BTS which the smartphone is currently attached to.





# Android: Localization essentials ...

- ❖ Cellular Localization is performed by recognizing the **mobile cell** where is located. HOW?

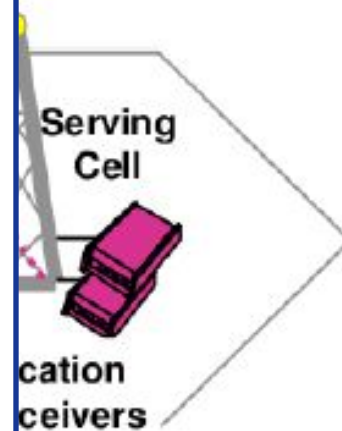
❖ Similar to previous case,

cu  
on  
ce  
sm  
att

Method	Accuracy
Cell-ID	10m-35km
Timing Advance (TA)	100m-550m
Angle of Arrival (AOA)	50m-150m
Uplink Time Of Arrival (U-TDOA)	50m-150m
Enhanced Observed Time Difference (E-OTD)	60m-200m
(Assisted-) GPS ((A)-GPS)	3m-10m



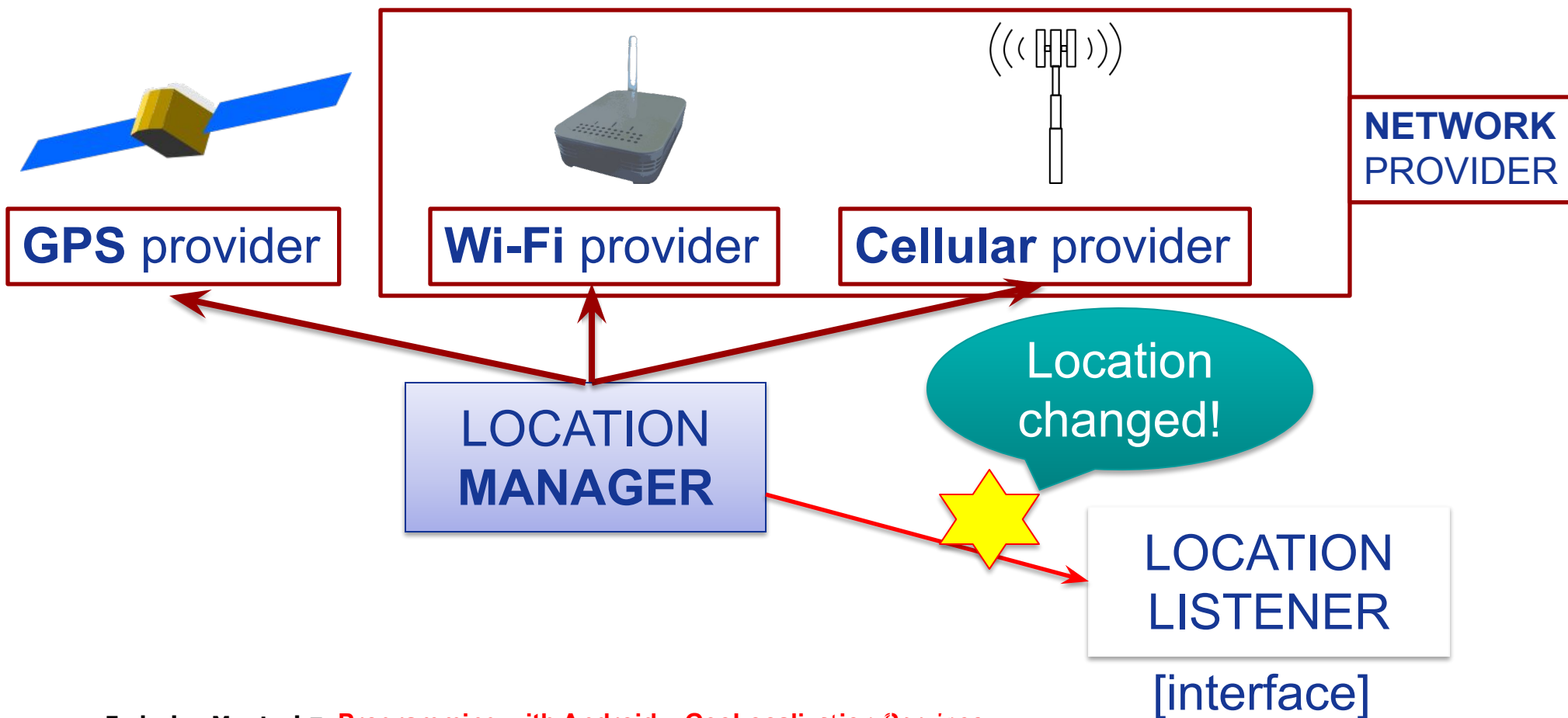
Location  
Calc & Control





# Android: Localization in Android

Q . HOW to retrieve the current position in Android?





# Android **Permission System**

- ❖ Up to 6.0 (excluded)
  - Just declare them in the manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

- ❖ Starting from 6.0
  - User can only grant a subset of the permission set
  - User can revoke permission after installing the app
  - Declare them in the manifest
    - And check if the permission is granted



# Android Location Permission

## ❖ ACCESS\_FINE\_LOCATION

- Allows the app to use any possible way to retrieve the location.

## ❖ ACCESS\_COARSE\_LOCATION

- Allows the app to use only location data coming from wifi / cellular localization.

## ❖ ACCESS\_BACKGROUND\_LOCATION

- To be requested in **addition** if you target API 29 or higher. Here is an elaborated article on how:

<https://developer.android.com/training/location/request-updates#request-background-location>



# Localization in Android

## ❖ Currently, two main systems

- android.location
- Location Services
  - Advised, easier and more efficient

## ❖ How?

- Main idea is to use a provider which returns the location
- With Location Services you use a FusedLocation provider which account for the device current status





# Android: Localization in Android

1. **Create a Location Listener object**, and implement the callback methods.

```
LocationListener locListener=new LocationListener() {  
  
    public void onLocationChanged(Location location) {  
    }  
    public void onStatusChanged(String provider, int status,  
Bundle extras) {  
    }  
    public void onProviderEnabled(String provider) {  
    }  
    public void onProviderDisabled(String provider) {  
    }  
}
```



# Android: Localization in Android

2. **Get a reference** to the Location Manager (system service).

```
LocationManager  
lm=(LocationManager)getService(Context.LOCATION_SERVICE)
```

3. **Register the LocationListener** in order to receive location updates from the Location Manager.

```
lm.requestLocationUpdates(provider, minTime, minDistance, locListener)
```

GPS\_PROVIDER  
NETWORK\_PROVIDER  
PASSIVE\_PROVIDER



# Android: Localization in Android

## 4. Add user permissions in the XML Manifest

```
<manifest>
  <uses-permissions
android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permissions
android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permissions android:name="android.permission.INTERNET"/>
</manifest>
```



# Android: Localization in Android

5. Get the **last known location** to reduce the latency caused by first location fix.

```
Location lastKnownLocation = locationManager.  
    getLastKnownLocation(locationProvider)
```

6. To save energy, **stop listening** to location updates when they are not needed anymore by the application.

```
locationManager.removeUpdates(locationListener)
```



# Android Location Based Services

- ❖ Added to make development easier
  - Introduces a FusedLocationProvider
- ❖ May also determine activity (more on this later ...)
- ❖ Background apps have their number of request reduced
- ❖ Need to add:

implementation 'com.google.android.gms:play-services-location:15.0.0'

- In the build.gradle file



# Location Services: **get the location**

## ❖ Obtain the FusedLocationProviderClient (onCreate)

```
FusedLocationProviderClient mFusedLocationClient =  
    LocationServices.getFusedLocationProviderClient(this);
```

## ❖ Get the location

```
mFusedLocationClient.getLastLocation()  
    .addOnSuccessListener(this, new OnSuccessListener<Location>() {  
        @Override  
        public void onSuccess(Location location) {  
            if (location != null) {  
                // do something  
            }  
        }  
    });
```



# Location Services: updates

## ❖ Create a LocationRequest

```
LocationRequest mLocationRequest = LocationRequest.create();  
mLocationRequest.setInterval(10000);  
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

❖ Location is then updated every 10 seconds

❖ Get it with `getLastLocation()`



# Location Services: update callback

- ❖ Some application may need a more continuous tracking

```
mLocationCallback = new LocationCallback() {  
    @Override  
    public void onLocationResult(LocationResult locationResult) {  
        for (Location location : locationResult.getLocations()) {  
            // Update UI with location data  
        }  
    }  
};
```

- ❖ Request updates

```
mFusedLocationClient.requestLocationUpdates(mLocationRequest,  
    mLocationCallback,  
    null /* Looper */); /* can use Looper.getMainLooper(); */
```





# Location Services: update callback

- ❖ Best practices also tell us to check the Location Settings to ensure the app will work no matter what
  - i.e. Sometimes Location Settings hinder the creation of a Location Request.
  
- ❖ More info at <https://developer.android.com/training/location/change-location-settings#get-settings>



# Android: Gmaps Important Dates ...

- **2004** □ Google Inc bought the Australian company *Where 2 Technologies*, that developed a prototype WebMap system.
- **2005** (February) □ Google Maps was announced
- **2006** □ Google Maps updated to use the same satellite image database as Google Earth
- **2007** □ Google Street View launched
- **2010** □ On Christmas and New Year's day, mobile usage of Google Maps surpassed desktop usage for the first time
- **NOW:** Google Maps, Google Sky, Google Moon, Google Mars, Google Transit, Google Aerial View, etc



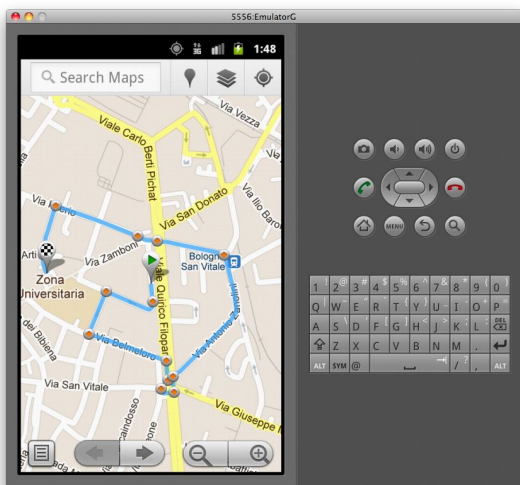
# Android: Gmaps Alternatives

- Since 2023 Google asks for credit card details to “prove that you’re not a robot”.
- Mapbox <https://docs.mapbox.com/android/maps/guides/>
- OsmDroid <https://github.com/osmdroid/osmdroid>



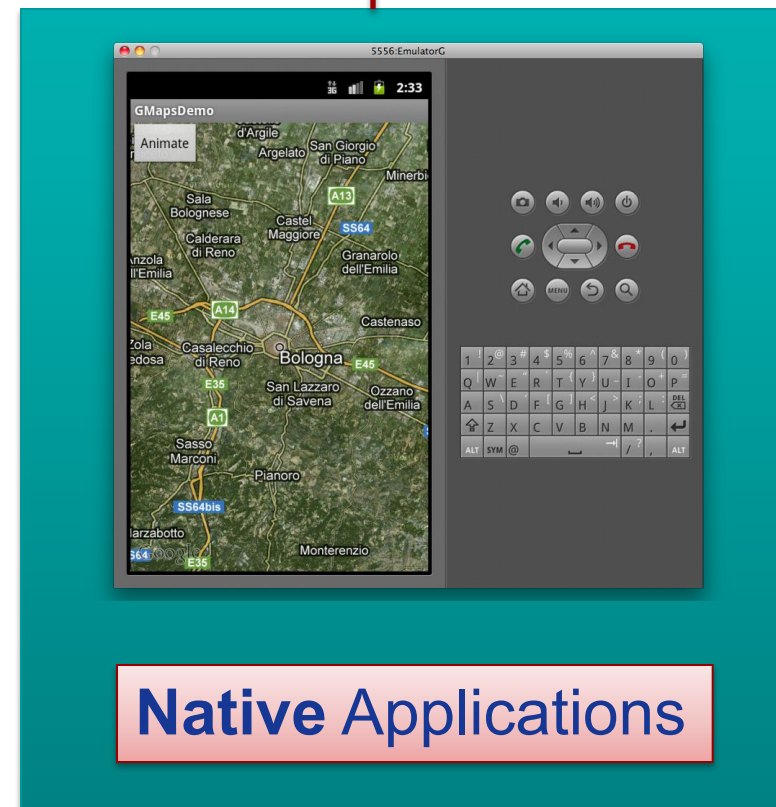
# Android: Deploying Map-based Apps

## Deploying Map-based Applications in Android



Hybrid Applications

WebView +  
Google Maps +  
Web technologies



Native Applications



# Android: Deploying Map-based Apps

## Two versions of Android Google Maps API

API v1



- Deprecated, not supported anymore since 18th March 2013.
- Still used for Android device with versions < 3.0 (unless API set is extended with support packages)

API v2

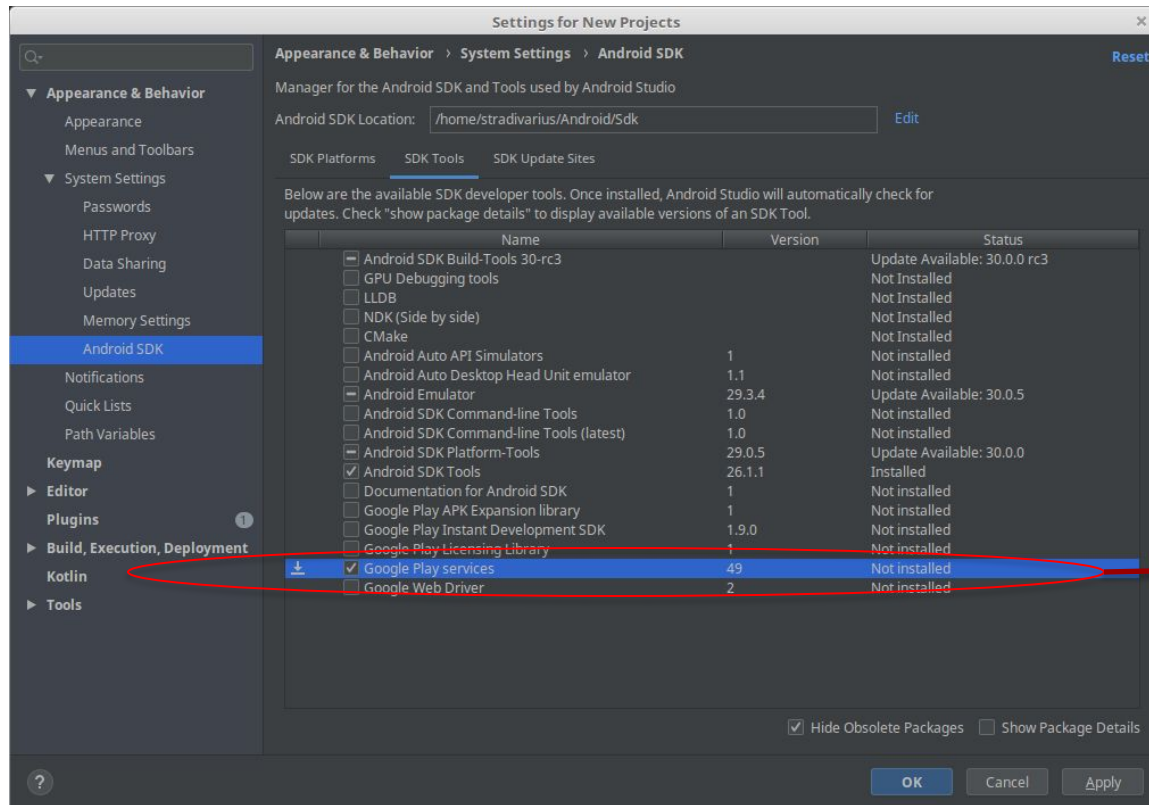


- Different installation procedures.
- Novel methods to insert a Map inside an Android app.
- Improved caching and visualization capabilities.



# Android: Installing Google APIs

## STEP -1: Install and Setup Google Play Service SDK



Tools  SDK Manager  
 SDK Tools

Check Google Play Services are **installed**, or **install** them otherwise

<https://developers.google.com/maps/documentation/android-sdk/start>



# Android: Getting a Google Play API Key

**STEP 0:** Get a valid Google Play **API Key** to utilize the Google Maps library.

**0.1:** Retrieve the fingerprint SHA1 of the certificate used to sign the apps.

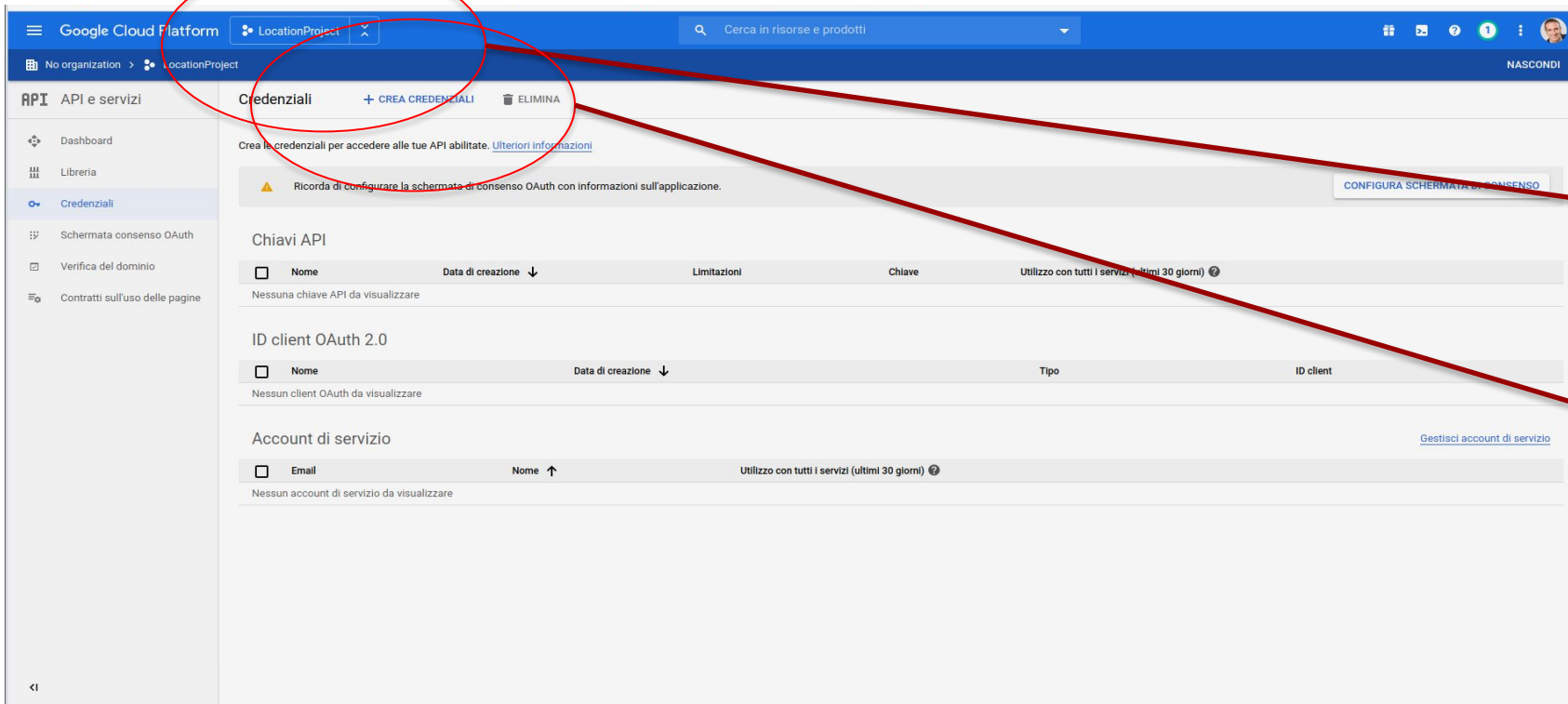
```
$ keytool -list -v -keystore  
~/.android/debug.keystore -storepass android -keypass  
android  
...  
androiddebugkey, Feb 1, 2020, PrivateKeyEntry,  
Certificate fingerprint (SHA1):  
A2:34:B1:A3:A5:BB:11:21:21:B3:20:56:92:12:AB:DB
```



# Android: Getting a Google Play API Key

**STEP 1:** Navigate with a browser to <https://cloud.google.com/console/google/maps-apis/overview>

**1.1:** Select the Google service you intend to use for your apps.



Create  
Entry for  
the project

Create the  
API KEY





# Android: Getting a Google Play API Key

**STEP 1:** Navigate with a browser to  
<https://cloud.google.com/console/google/maps-apis/overview>

## Restrict:

- Restrict the key to Android Applications
- Insert the SHA1 Key, and the package name:

BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;  
**it.unibo.stradivarius.mapsapp**

- Restrict to Maps API (if not listed, you need to enable it from your home)
- **For each application/package**  **get a new Activation Key.**



# Android: **Google MAPs library overview**

What can I do with Google MAPs v2 library in Android?

- 1. Integrate** a Google Map into an Android application
- 2. Manage** the camera
- 3. Add** information layers to the Map
- 4. Manage** user events



# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

- Internet Access
- Localization capabilities
- Access to Google Web services
- OpenGL ES version 2 libraries
- Access to network state



# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<meta-data
  android:name="com.google.android.geo.API_KEY"
  android:value="API_activation_key" />
```

```
<permission
  android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
  android:protectionLevel="signature" />
<uses-permission
  android:name="com.example.mapdemo.permission.MAPS_RECEIVE" />
<uses-feature
  android:glEsVersion="0x00020000"
  android:required="true" />
```



# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<meta-data
  android:name="com.google.android.geo.API_KEY"
  android:value="API_activation_key" />
```

Specifically for the **Android Studio** projects:

```
<meta-data
  android:name="com.google.android.gms.version"
  android:value="@integer/
    google_play_services_version" />
```



# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```



# Android: Inserting a Map inside the App

In order to insert a Google Map into a mobile Application:

- Add a **MapFragment** (in androidx SupportMapFragment) to the current Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



# Android: Inserting a Map inside the App

A MapFragment is a container of the **GoogleMap** object, which is a View containing the map and managing the events.

```
private GoogleMap mMap;  
...  
mMap = ((SupportMapFragment) getSupportFragmentManager()  
        .findFragmentById(R.id.map)).getMap();
```

Differences with **Android Maps v1 libs**:

- No need to use a MapActivity, use a regular Activity instead.
- Improved caching and drawing functionalities.





# Android: Inserting a Map inside the App

The correct handling however includes callbacks:

```
@Override
public void onMapReady(GoogleMap map) {
    ...
}
```

Activity implements OnMapReadyCallback ...

```
SupportMapFragment mapFragment = (SupportMapFragment)
    getSupportFragmentManager()
        .findFragmentById(R.id.map);
mapFragment.getMapAsync(this); // fires the callback
```



# Android: **Customize** the Map

## How to customize the Google Map?

- Define the **Map type**, governing the overall representation of the map

```
nMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

**Normal**  Typical road map.

**Hybrid**  Satellite photograph data with road maps added.

**Satellite**  Satellite photograph data. Road and feature labels are not visible.

**Terrain**  Topographic data. The map includes colors, contour lines and labels, and perspective shading.

**None**  no tiles, empty grid.



# Android: **Customize** the Map

The **LatLng** class allows to define a point on the map, expressed through the latitude/longitude coordinates.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781, 11.356387);
```

```
private static final LatLng FLORENCE_POINT = new  
LatLng(43.771373, 11.248069);
```

**LatLng** class (API v2)  **Geopoint** class (API v1)



# Android: **Customize** the Map

Q. How to customize the Google Map?

A. Define the **properties of the Camera** applied to the Map.

**Location**  expressed in forms of latitude/longitude coordinates.

**Zoom**  defines the scale levels of the map.

**Bearing**  defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.

**Tilt**  viewing angle, measured as degrees from the nadir.



# Android: **Customize** the Map

Camera properties can be set individually, or collectively through the **CameraPosition** object.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781, 11.356387);
```

```
CameraPosition cameraPosition = new CameraPosition.  
Builder()  
.target(BOLOGNA_POINT)  
.zoom(17)  
.bearing(90)  
.tilt(30)  
.build();
```



# Android: **Customize** the Map

Two methods to modify the position of the camera:

```
mMap.moveCamera(cameraPosition);
```

- Update the camera properties immediately.

```
mMap.animateCamera(cameraPosition);
```

```
mMap.animateCamera(cameraPosition, duration, call);
```

- Update the camera properties through an animation, eventually adding a delay and a callback to be invoked when the animation stops.

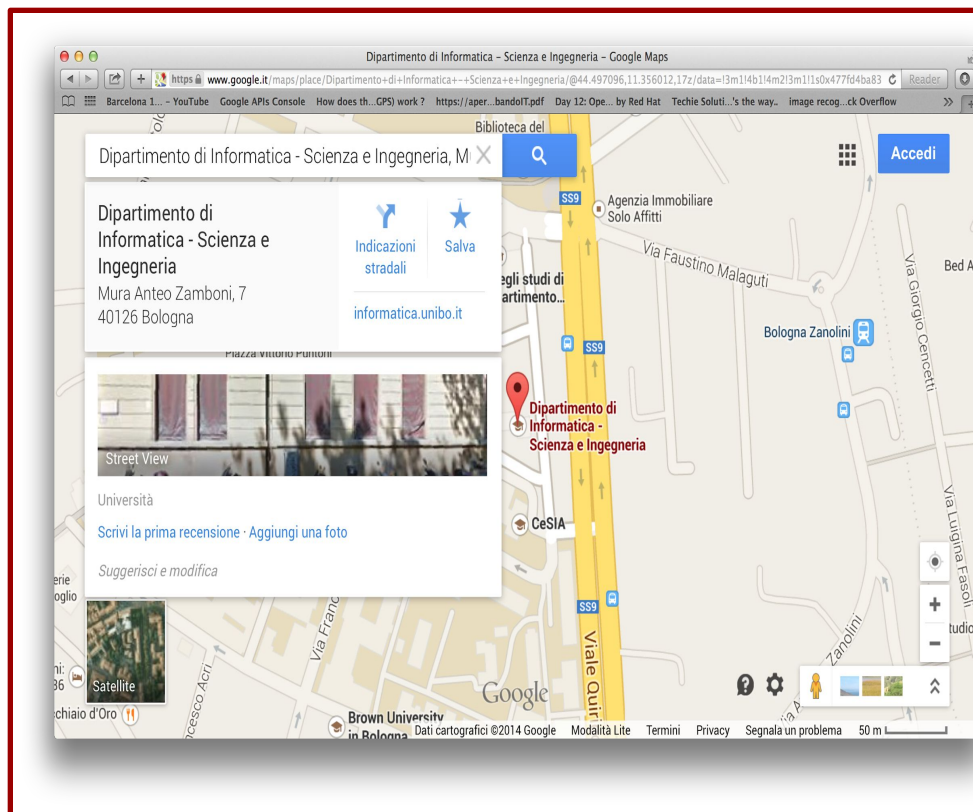


# Android: **Customize** the Map

**Markers** can be used to identify locations on the GoogleMap.

Markers can be customized in terms of:

- ❑ **Icon** to be displayed
- ❑ **Position** of the marker on the map
- ❑ **Title** and text to be displayed
- ❑ **Events** to be managed





# Android: **Customize** the Map

**Markers** can be used to identify locations on the GoogleMap.

**position**  Lat/Long coordinates

**title**  string displayed in the info window when the user taps the marker

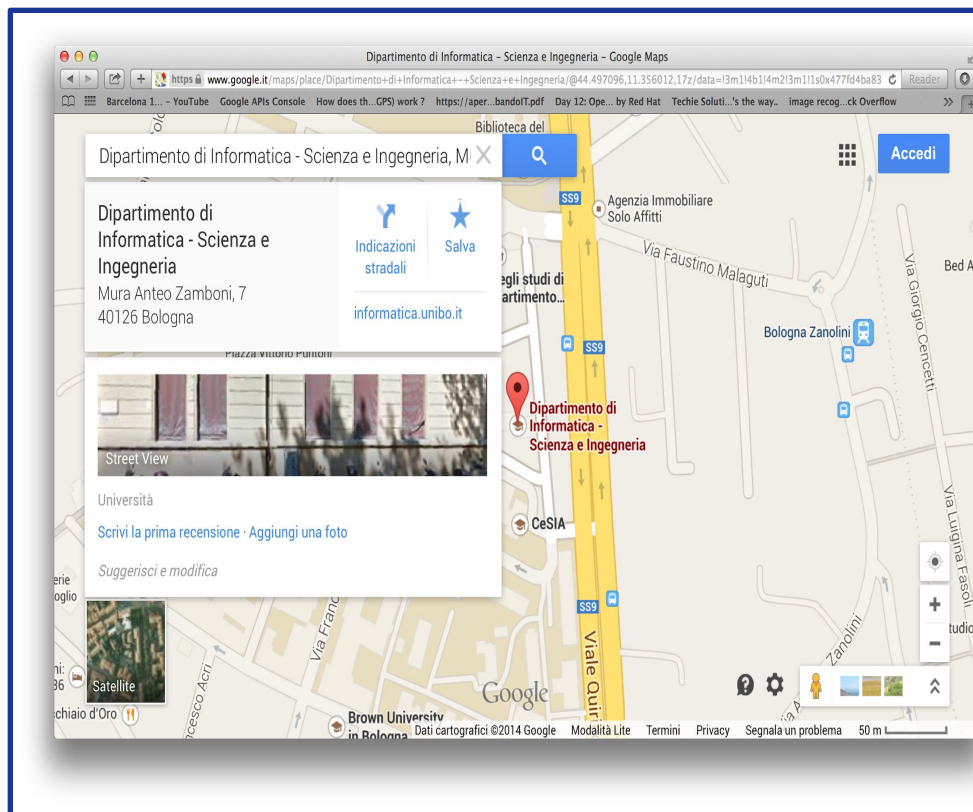
**snippet**  additional text in the info window

**icon**  image/color of the marker

**alpha**  opacity of the marker

**draggable**  (true/false)

**visible**  (true/false)







# Android: **Customize** the Map

**Markers** can be used to identify locations on the GoogleMap.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781, 11.356387);
```

```
Marker bologna =  
myMap.addMarker(newMarkerOptions().position(BOLOGNA_POIN  
T));
```

```
Marker bologna= mMap.addMarker(new MarkerOptions()  
    .position(Bologna)  
    .title("Bologna downtown")  
    .snippet("Visit the city centre"));
```



# Android: **Customize** the Map

**Markers** can be used to identify locations on the GoogleMap.

**EVENTS** associated to a Marker:

**ClickEvents**  implement the `OnMarkerClickListener` interface, and the `onMarkerClick(Marker)` method.

**DragEvents**  implement the `OnMarkerDragListener` interface, and the `onMarkerDragEnd(Marker)` method.

**InfoWindow Click Events**  implement the `OnInfoWindowClickListener` interface, and the `onInfoWindowClick(Marker)` method.



# Android: **Customize** the Map

Developers can handle the **events** on the Google Map.

Events are managed through the **listener mechanism** seen so far ...

**CLICK** events  Implement the `OnMapClickListener` interface and the `OnMapLongClickListener` method.

**CAMERA** events  Implement the `OnCameraChangeListener` interface and the `onCameraChange(CameraPosition)` method.



# Android: **Customize** the Map

Developers can handle the **events** on the Google Map.

```
public class MainActivity extends Activity
    implements OnMapClickListener {
    private GoogleMap mMap;

    protected void onCreate(Bundle savedInstanceState) {
    ...
    mMap.setOnMapClickListener(this);
    ...
    }

    public void onMapClick(LatLng position) {
        // Handle the click events here ...
    }
}
```



# Android: **Customize** the Map

**Shapes** can be used to identify sections of the GoogleMap.

- Polylines**  define a set of LatLong objects, and connect them through a set of lines. Possible to define the stroke and colors of the lines.
- Polygons**  define a set of LatLong objects, and connect them through a closed polygon. Possible to define the stroke and colors of the lines.
- Circles**  define a LatLong object and a radius, and draw a circle centered at the point. Define pen color/stroke as above.



# Android: **Customize** the Map

**Shapes** can be used to identify sections of the GoogleMap.

```
PolygonOptions rectOptions = new PolygonOptions()  
    .add(BOLOGNA_P1)  
    .add(BOLOGNA_P2)  
    .add(BOLOGNA_P3);
```

```
Polygon polyline = mMap.addPolygon(rectOptions);
```

```
CircleOptions circleOptions = new CircleOptions()  
    .center(BOLOGNA_P1)  
    .radius(1000)  
    .strokeColor(Color.RED);
```

```
Circle circle = mMap.addCircle(circleOptions);
```



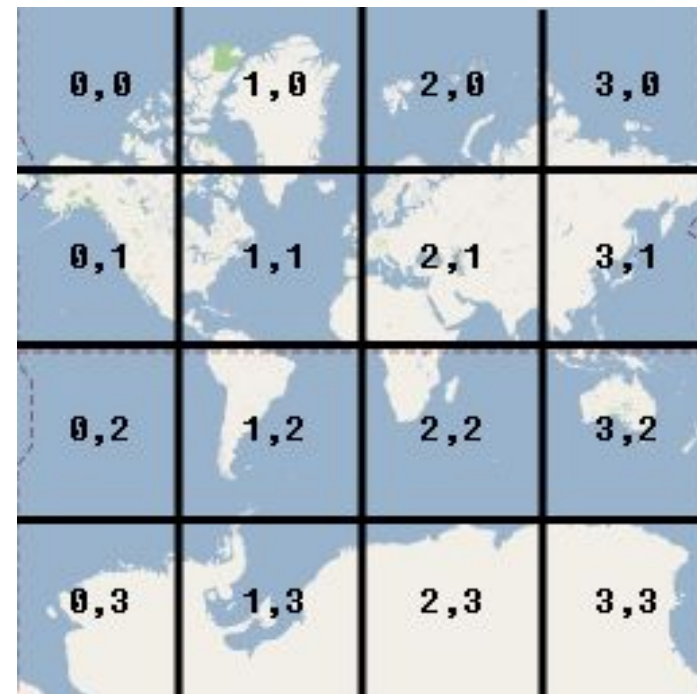
# Android: **Customize** the Map

## MapOverlays and Tiling

```
TileProvider tileProvider = new
UrlTileProvider(256, 256) {

    @Override
    public URL getTileUrl(int x,
int y, int zoom) { ... }
}

TileOverlay tileOverlay =
map.addTileOverlay(new
TileOverlayOptions()
    .tileProvider(tileProvider));
```



<https://developers.google.com/maps/documentation/android-sdk/tileoverlay?hl=en>



# Android: **Customize** the Map

**Google Direction API** □ services that calculates directions between a source and a destination, including different transportations modes (e.g. driving, walking, biking).

- ✦ **Remote Service:** send an HTTP request and get an HTTP reply
- ✦ **Activation key** needed from the Google API Console
- ✦ Subject to **usage limits:** 2500 directions/day for free API, 100000 directions/day for business API (8 vs 24 waypoints)
- ✦ Direction API data must be displayed on the Map
- ✦ It is not designed to support real-time navigation applications





# Android: **Customize** the Map

- Direction API requests takes the following form:

<http://maps.googleapis.com/maps/api/directions/output?parameters>

**JSON** (recommended) or **XML**

## REQUIRED

**origin** □ latitude/longitude coordinates or address (**geocoding** performed)

**destination** □ latitude/longitude coordinates or address

**sensor** □ request comes from a device with location sensor (true/false)

**key** □ **API Key** of the Google Direction Service



# Android: **Customize** the Map

- Direction API requests takes the following form:

<https://maps.googleapis.com/maps/api/directions/output?parameters>

## OPTIONAL

- mode** □ transportation mode (driving, walking, bicycling, transit)
- waypoints** □ array of waypoints which must appear on the route
- alternatives** □ (true/false) decide to show single or multiple routes
- avoid** □ avoid specific features (tolls, highways, ferries)
- departure\_time** □ desired time of departure
- arrival\_time** □ desired time of arrival
- language** □ language of the results (e.g. route indications)



# Android: **Customize** the Map

- Example of Google Direction requests

```
https://maps.googleapis.com/maps/api/directions/json?origin=Bologna&destination=Modena&sensor=false&key={API_KEY}
```

```
https://maps.googleapis.com/maps/api/directions/json?origin=Bologna&destination=Modena&sensor=false&key={API_KEY}&avoid=highways  
&mode=transit
```

```
https://maps.googleapis.com/maps/api/directions/json?origin=Bologna&destination=Modena&waypoints=Vignola|Maranello&sensor=false&  
key={API_KEY}&avoid=highways&mode=transit
```



# Android: Customize the Map

```
{
  "status": "OK",
  "routes": [ {
    "summary": "I-40 W",
    "legs": [ {
      "steps": [ {
        "travel_mode":
"DRIVING",
        "start_location": {
          "lat": 41.8507300,
          "lng": -87.6512600
        },
        "end_location": {
          "lat": 41.8525800,
          "lng": -87.6514100
        },

```

JSON result of the query

```
"polyline": {
  "points": "a~l~Fjk~uOwHJy@P"
},
"duration": {
  "value": 19,
  "text": "1 min"
}
"html_instructions": "Head Morgan St",
"distance": {
  "value": 207,
  "text": "0.1 mi"
}
},
.....
```



# Android: Google Maps library overview

**GeoCoding** □ Technique to convert an Address into a Geo (lat/long) point, or viceversa (reverse geocoding)...

Implemented by the Geocoder class

```
public Geocoder(Context context)
```

Main methods:

- `public List<Address> getFromLocation(double latitude, double longitude, int maxResults)`
- `public List<Address> getFromLocationName(String locationName, int maxResults)`



# GeoCoder Example

## ❖ getFromLocation

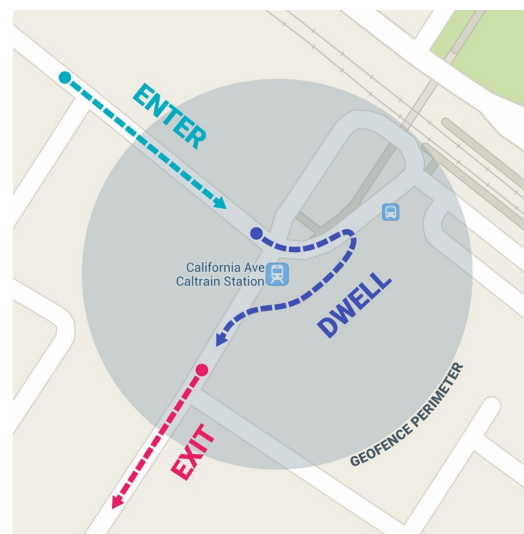
```
addresses = geocoder.getFromLocation(  
    location.getLatitude(),  
    location.getLongitude(),  
    1);
```

- ❖ Each coordinate may have multiple addresses
  - Specify the addresses you want with the third parameter



# Geofencing

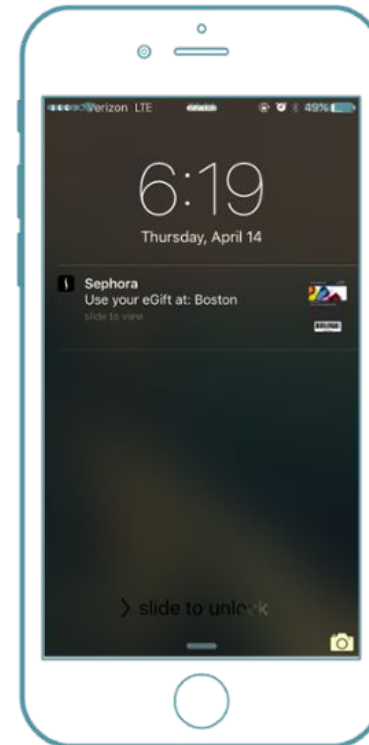
- ❖ Sometimes your app tracks the user to retrieve the path
- ❖ But it may also track it to understand when the user enters/stays/exits a certain area
- ❖ Solution 1: polling
- ❖ Solution 2: Geofencing
  - Technique which creates geo boundaries





# Geofencing Examples

- ❖ Proximity Marketing
- ❖ Smart Home optimization
- ❖ Safety
- ❖ Social networking
- ❖ Smart calendar







# Geofencing: **basics**

- ❖ Combines user location with proximity
  - Specify latitude-longitude-radius
- ❖ Can have multiple geofences
  - Limit of 100
  - Can configure Location Services to inform you about events
  - Geofences also have an expiration time
- ❖ Need `ACCESS_FINE_LOCATION`
  - `ACCESS_BACKGROUND_LOCATION` since Android 10



# Geofencing: **example**

## ❖ Get the GeofencingClient

```
mGeofencingClient = LocationServices.getGeofencingClient(this);
```

## ❖ Create a list and add geofences

```
mGeofenceList.add(new Geofence.Builder()  
    .setRequestId(myID)  
    .setCircularRegion(myLatitude, myLongitude, myRadius)  
    .setExpirationDuration(myExpirationInMilliseconds)  
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |  
        Geofence.GEOFENCE_TRANSITION_EXIT)  
    .build());
```

```
GeofencingRequest.Builder builder = new GeofencingRequest.Builder();  
builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);  
builder.addGeofences(mGeofenceList);  
builder.build();
```



# Geofencing: **example**

## ❖ How to react? Specify what to launch with a PendingIntent

```
geofencingClient.addGeofences(geofencingRequest, geofencePendingIntent)
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
        }
    });
```

```
GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
```



# Geofencing: best practices

- ❖ Battery efficiency
  - Use a higher `setNotificationResponsiveness`
  - Use a larger radius
- ❖ Reduce number of alerts
  - Explore `GEOFENCE_TRANSITION_DWELL`
  - Set a reasonable loitering delay
- ❖ Geofences are destroyed when they expire
  - Re-register them only if needed